# Gibb's sampling for Linear Regression

August 2, 2020

## 1 Introduction

In most of the real-world probabilistic models, exact inference of the parameters is intractable. Then sampling algorithms are utilised to obtain some parameter approximations. Such inference algorithms based on numerical sampling are called Markov Chain Monte Carlo (MCMC) techniques.

Sampling from a distribution provides a window onto its properties, which allows us to understand its nature. It allows us to estimate those characteristics – the mean, variance and quantiles – that we usually want as outputs from a Bayesian analysis.

***A Student's Guide to Bayesian Statistics - Ben Lambert*** (1)

Here, we will look at one such algorithm called Gibb's sampling.

Let $p(x)$ be the target distribution where $x = (x_1, ..., x_d)$. Gibbs sampling consists of a random walk on an undirected graph whose vertices correspond to the values of $x = (x_1, ..., x_d)$ and in which there is an edge from $x$ to $y$ if $x$ and $y$ differ in only one coordinate. Thus, the underlying graph is like a $d$-dimensional lattice except that the vertices in the same coordinate line form a clique.

***Foundations of Data Science - Avrim Blum, John Hopcroft,and Ravindran Kannan*** (2)

The calculation of the posterior may not be possible but it is sometimes possible to find the conditional probabilities of the parameters. Gibb's sampling updates each parameter by sampling from its conditional distribution given the other parameters.

Gibb's sampling is more efficient than Metropolis-Hastings algorithm as it accepts all the samples and rejects none.

If we have to create a map of the underground iron deposits and do not have the resources to look at every single point in the entire desert, then, Gibb's sampling says we start in a random location in the desert, we can use the satellite to determine our next sampling point in the east–west direction. We then move to the new point and use our satellite to determine a random point in the north–south direction and move there. We continue this process, alternating between moving to a random location in the east-west and then north-south directions. This method is the approach used by Gibb's sampling.
We notice two differences between the moves selected by these samplers. First, when using the Metropolis sampler, we often reject proposed steps and remain at our previous location in a given iteration. This contrasts with the Gibbs sampler, where we always move to a new spot in each iteration. Second, for the Metropolis sampler, we change both our north and east coordinates simultaneously, whereas in the Gibbs case we move along only one direction at a time.

(1)

Gibb's sampling for a model with 2 variables $\theta = (\theta_1, \theta_2)$ and data $x$ consists of the following steps:
Chose a random starting point $(\theta_1^0, \theta_2^0)$, then iterate over the following:

1. Chose a parameter update order - $(\theta_1, \theta_2)$ or $(\theta_2, \theta_1)$.
2. In the order chosen in step 1, sample from the conditional parameter of the each parameter with the other updated parameters. So, for the chosen order $(\theta_1, \theta_2)$, first, we sample from $p(\theta_1^1|\theta_2^0, x)$ and then sample from $p(\theta_2^1|\theta_2^1, x)$.

Continue this until convergence.

Let us now try to solve a simple Linear Regression problem with Gibb's sampling.

## 2 Bayesian Linear Regression

Here we have a normal Linear Regression (LR):

$$y_i \sim \mathcal{N}(\beta_0 + \beta_1 x_i, 1/\tau), i = 1, ..., N$$

We are interested in finding the posterior distribution of $\beta_0$ (intercept), $\beta_1$ (slope) and $\tau$ (inverse of the variance called precision). Assuming $N$ i.i.d observations, the likelihood of this model is,

$$\begin{aligned} p(y, x|\beta_0, \beta_1, \tau) &= L(y_1, ..., y_N, x_1, ..., x_N|\beta_0, \beta_1, \tau) \\ &= \prod_{i=1}^{N} \mathcal{N}(\beta_0 + \beta_1 x, 1/\tau) \end{aligned}$$

Let us define conjugate priors of the parameters:
1. $\beta_0 \sim \mathcal{N}(\mu_0, 1/\tau_0)$ 2. $\beta_1 \sim \mathcal{N}(\mu_1, 1/\tau_1)$ 3. $1/\tau \sim Gamma(\alpha, \beta)$

Now, let us define these in python.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Let us now derive the updates for $\beta_0$ and $\beta_1$
The general approach is the following:

1. Write down the posterior conditional density in log-form.
2. Discard all the terms that don't depend on the current sampling variable.
3. Now, assume this is the density for our sampling variable and all other variables are fixed.
4. Find out the distribution of this log-density and that is the conditional sampling density we want.

Before we derive the updates, there is one important derivation that we need to be aware of. Let us assume there is variable $x$ that is normally distributed with mean $\mu$ and precision $\tau$,

$$\begin{aligned}
f(x|\mu, 1/\tau) &= \sqrt{\frac{\tau}{2\pi}}\, e^{-\frac{\tau}{2}(x-\mu)^2} \\
&= \text{const} \cdot\ e^{(-\frac{\tau x^2}{2} - \frac{\tau\mu^2}{2} - \tau x\mu)} \\
ln(f(x|\mu, 1/\tau)) &= ln(\text{const}) + ln(e^{(-\frac{\tau x^2}{2} - \frac{\tau\mu^2}{2} - \tau x\mu)}) \\
&= ln(\text{const}) - \frac{\tau x^2}{2} - \frac{\tau\mu^2}{2} + \tau x\mu \\
&= \frac{\tau x^2}{2} + \tau x\mu \quad \text{(drop terms that are independent of x)}
\end{aligned}$$

Having derived this, note that,

1. The coefficient of $x^2$ is $\frac{-\tau}{2}$
2. The coefficient of $x$ is $\tau\mu$

## 2.1 Deriving update for $\beta_0$

$$\begin{aligned}
p(\beta_0|\beta_1, \tau, y, x) &\propto p(y, x|\beta_0, \beta_1, \tau)p(\beta_0) \\
&\propto \prod_{i=1}^{N} \mathcal{N}(\beta_0 + \beta_1 x_i, 1/\tau) \cdot \mathcal{N}(\mu_0, 1/\tau_0) \\
ln(p(\beta_0|\beta_1, \tau, y, x)) &\propto \sum_{i=1}^{N} ln(\mathcal{N}(\beta_0 + \beta_1 x_i, 1/\tau)) + ln(\mathcal{N}(\mu_0, 1/\tau_0))
\end{aligned}$$

Expanding the first term on the RHS,

$$\begin{aligned}
\sum_{i=1}^{N} ln(\mathcal{N}(\beta_0 + \beta_1 x_i, 1/\tau)) &= \sum_{i=1}^{N} ln[\sqrt{\frac{\tau}{2\pi}} e^{-\frac{\tau}{2}(y_i - (\beta_0 + \beta_1 x_i))^2}] \\
&= \sum_{i=1}^{N} (ln[\sqrt{\frac{\tau}{2\pi}}] + ln[e^{-\frac{\tau y_i^2}{2} - \frac{\tau(\beta_0 + \beta_1 x_i)^2}{2} + \tau y_i(\beta_0 + \beta_1 x_i)}]) \\
&= \sum_{i=1}^{N} (ln[\sqrt{\frac{\tau}{2\pi}}] + [-\frac{\tau y_i^2}{2} - \frac{\tau\beta_0^2}{2} - \frac{\tau\beta_1^2 x_i^2}{2} - \tau\beta_0\beta_1 x_i + \tau y_i\beta_0 + \tau y_i\beta_1 x_i]) \\
&\quad \text{(drop all the terms that do not contain } \beta_0\text{)} \\
&= \sum_{i=1}^{N} (-\frac{\tau\beta_0^2}{2} - \tau\beta_0\beta_1 x_i + \tau y_i\beta_0)
\end{aligned}$$

Expanding the first term on the RHS,

$$\begin{aligned} ln(\mathcal{N}(\mu_0, 1/\tau_0)) &= ln(\sqrt{\frac{\tau_0}{2\pi}}) + ln(e^{-\frac{\tau_0}{2}(\beta_0-\mu_0))^2}) \\ &= ln(\sqrt{\frac{\tau_0}{2\pi}}) + (-\frac{\tau_0\beta_0^2}{2} - \frac{\tau_0\mu_0^2}{2} + \tau_0\beta_0\mu_0)) \\ &\quad \text{(drop all the terms that do not contain } \beta_0\text{)} \\ &= -\frac{\tau_0\beta_0^2}{2} + \tau_0\beta_0\mu_0 \end{aligned}$$

Having derived this,

1. The coefficient of $\beta_0^2$ is $\frac{-\tau N}{2} - \frac{\tau_0}{2}$
2. The coefficient of $\beta_0$ is $\tau\sum_{i=1}^{N}(y_i - \beta_1 x_i) + \tau_0\mu_0$

So, $\beta_0$ looks Gaussian and matching the coefficients of $\beta_0$ with that of variable $x$ derivation from earlier and calculating the mean and precision, we define the conditional density of $\beta_0$ as,

$$\sim \mathcal{N}(\frac{\tau_0\mu_0 + \tau\sum_{i=1}^{N}(y_i - \beta_1 x_i)}{\tau_0 + \tau N}, \frac{1}{\tau_0 + \tau N})$$

Let us implement the update step in python.

```
[2]: def sample_beta_0(y,x,beta_1,tau,mu_0,tau_0):
         N = len(y)
         assert len(x) == N

         precision = tau_0 + tau*N
         mean      = (tau_0*mu_0 + tau*np.sum(y - beta_1*x)) / precision

         return np.random.normal(mean, 1/np.sqrt(precision))
```

Great progress!
Now, let us find the conditional density of $\beta_1$.

## 2.2 Deriving update for $\beta_1$

This derivation is very much similar to that of $\beta_0$. So, go ahead and do it yourself! It is good practice.

$$\begin{aligned} p(\beta_1|\beta_0, \tau, y, x) &\propto p(y, x|\beta_0, \beta_1, \tau)p(\beta_1) \\ &\propto \frac{-\tau_1\beta_1^2}{2} - \frac{\tau\sum_{i=1}^{N}\beta_1^2 x_i^2}{2} + \tau_1\beta_1\mu_1 + \tau\beta_1\sum_{i=1}^{N} x_i(y_i - \beta_0) \end{aligned}$$

Having derived this,

1. The coefficient of $\beta_1^2$ is $\frac{-\tau_1}{2} - \frac{\tau\sum_{i=1}^{N} x_i^2}{2}$
2. The coefficient of $\beta_1$ is $\tau\sum_{i=1}^{N} x_i(y_i - \beta_0) + \tau_1\mu_1$

So, $\beta_1$ looks Gaussian and matching the coefficients of $\beta_1$ with that of variable $x$ derivation from earlier and calculating the mean and precision, we define the conditional density of $\beta_1$ as,

$$\sim \mathcal{N}(\frac{\tau_1\mu_1 + \tau\sum_{i=1}^N x_i(y_i - \beta_0)}{\tau_1 + \tau\sum_{i=1}^N x_i^2}, \frac{1}{\tau_1 + \tau\sum_{i=1}^N x_i^2})$$

Let us implement the update step in python.

[3]:
```
def sample_beta_1(y,x,beta_0,tau,mu_1,tau_1):
    N = len(y)
    assert len(x) == N

    precision = tau_1 + tau*np.sum(x*x)
    mean      = (tau_1*mu_1 + tau*np.sum(x*(y - beta_0))) / precision

    return np.random.normal(mean, 1/np.sqrt(precision))
```

Now, onto one last derivation.

## 2.3 Deriving update for $\tau$

Since this is non-gaussian, let us again derive a Gamma function with variable $x$.

$$\begin{aligned} p(x|\alpha,\beta) &= \frac{1}{\Gamma}\beta^\alpha x^{\alpha-1}e^{-\beta x} \\ ln(p(x|\alpha,\beta)) &= ln(\frac{1}{\Gamma(\alpha)}) + \alpha ln(\beta) + (\alpha-1)ln(x) - \beta x \\ &\quad \text{(drop all the terms that do not contain } x\text{)} \\ &\propto (\alpha-1)ln(x) - \beta x \end{aligned}$$

Having derived this,

1. The coefficient of $\tau$ is $-\beta$
2. The coefficient of $ln(\tau)$ is $\alpha - 1$

Now, we want,

$$\begin{aligned} p(\tau|\beta_0,\beta_1,y,x) &\propto p(y,x|\beta_0,\beta_1,\tau)p(\tau) \\ &\propto \prod_{i=1}^N \mathcal{N}(y_i|\beta_0+\beta_1 x_i, 1/\tau)\cdot p(x|\alpha,\beta) \\ ln(p(\tau|\beta_0,\beta_1,y,x)) &\propto \sum_{i=1}^N ln[\sqrt{\frac{\tau}{2\pi}}e^{\frac{-\tau}{2}(y_i-(\beta_0+\beta_1x_i))^2}] + ln[\frac{1}{\Gamma(\alpha)}\beta^\alpha\tau^{\alpha-1}e^{-\beta\tau}] \\ &\quad \text{(drop all the terms that do not contain } \tau\text{)} \\ &\propto \sum_{i=1}^N(\frac{1}{2}ln(\tau) - \frac{\tau}{2}(y_i-(\beta_0+\beta_1x_i))^2) + (\alpha-1)ln(\tau) - \beta\tau \\ &\propto \frac{N}{2}ln(\tau) - \frac{\tau}{2}\sum_{i=1}^N(y_i-\beta_0-\beta_1x_i)^2 + (\alpha-1)ln(\tau) - \beta\tau \end{aligned}$$

Making the coefficient like earlier, we get, 1. The coefficient of $\tau$ is $-\frac{\tau}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \beta_1 x_i)^2 - \beta$ 2. The coefficient of $ln(\tau)$ is $\frac{N}{2} + \alpha - 1$

The conditional density of the $\tau$ update step is,

$$\sim Ga(\alpha + \frac{N}{2}, \beta + \frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \beta_1 x_i)^2)$$

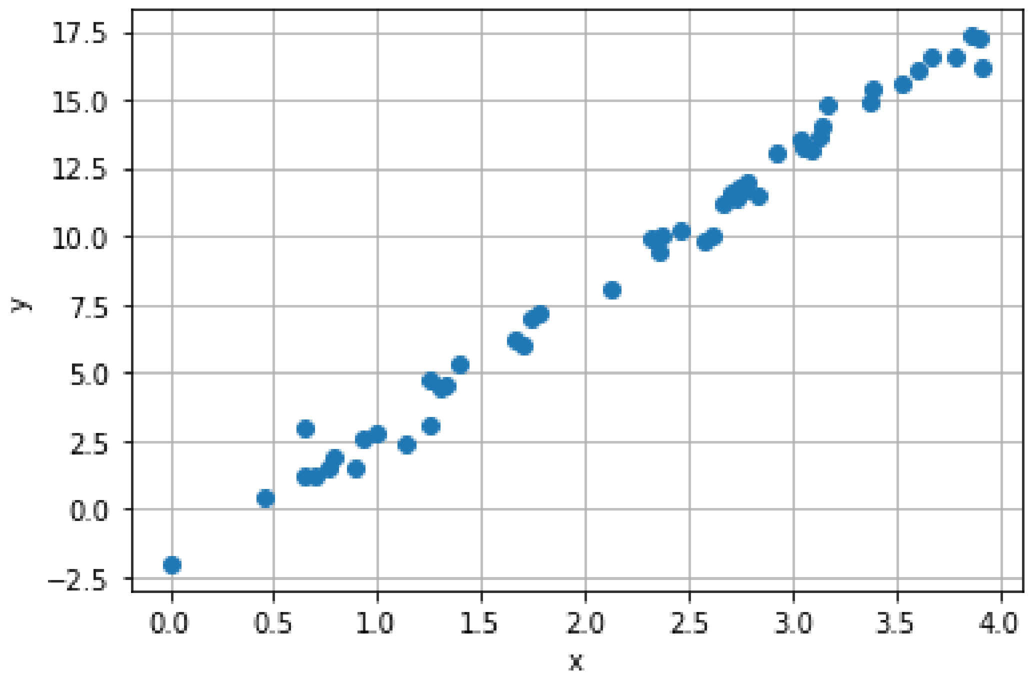
[4]:
```
def sample_tau(y,x,beta_0,beta_1,alpha,beta):
    N = len(y)
    assert len(x) == N

    alpha_new = alpha + N/2
    beta_new  = beta + np.sum((y - beta_0 - beta_1*x)**2)/2

    return np.random.gamma(alpha_new,1/beta_new)
```

All the derivations are done!

## 2.4 Setup a toy example

Let us create a toy example to test the functions. Let
$\beta_0 = -2$,
$\beta_1 = 5$,
$\tau = 3$.

[5]:
```
beta_0_true = -2
beta_1_true = 5
tau_true    = 3

N = 50
x = np.random.uniform(0,4,N)
y = np.random.normal(beta_0_true+beta_1_true*x,1/np.sqrt(tau_true))

plt.figure()
synth_plot = plt.plot(x,y,"o")
plt.xlabel("x")
plt.ylabel("y")
plt.grid()
plt.show()
```



## 2.5 Onto defining the Gibb's sampler itself

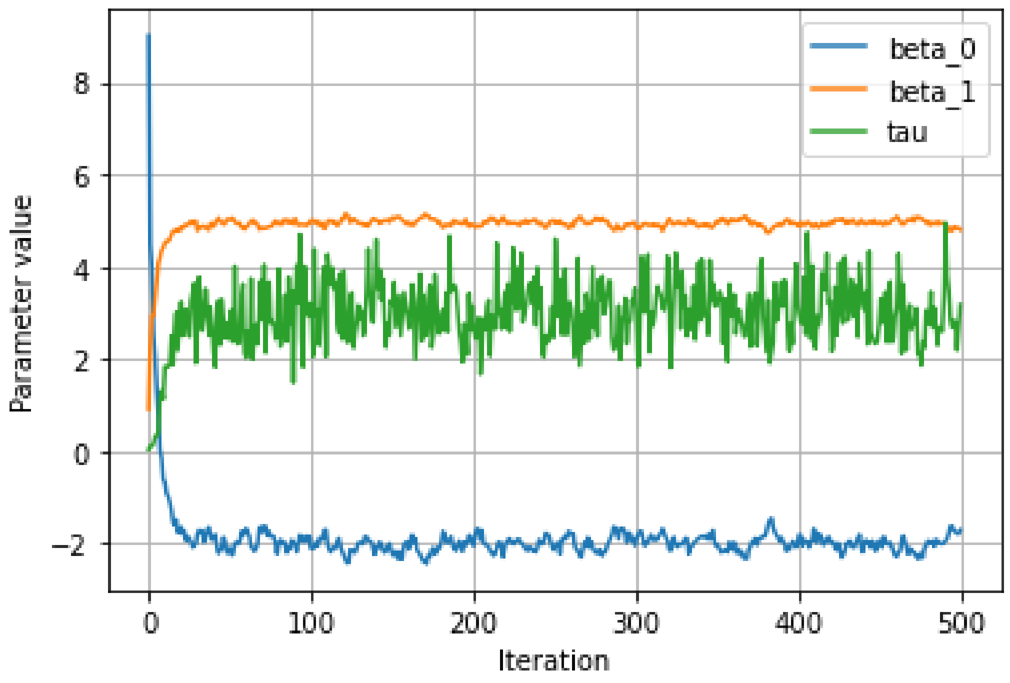
Let us first specify a random initial values for the parameters and define specific hyperparameters, like, $\beta_0$ and $\beta_1$ can be $\mathcal{N}(0,1)$ and $\tau$ can be $Gamma(2,1)$.
We can then define the Gibb's sampler according to the update steps mentioned before.

[6]:
```
# Specify initial values
init = {"beta_0": 0,
        "beta_1": 0,
        "tau": 2}

# Specify hyper parameters
hypers = {"mu_0": 0,
         "tau_0": 1,
         "mu_1": 0,
         "tau_1": 1,
         "alpha": 2,
         "beta": 1}
```

[7]:
```
def gibbs(y, x, iters, init, hypers):
    assert len(y) == len(x)
    beta_0 = init["beta_0"]
    beta_1 = init["beta_1"]
    tau = init["tau"]

    store_param = np.zeros((iters, 3))

    for i in range(iters):

        # Update beta_0 with other old parameters
        beta_0 = sample_beta_0(y, x, beta_1, tau, hypers["mu_0"],␣
 ↪hypers["tau_0"])

        # Update beta_1 with new beta_0 and old tau
        beta_1 = sample_beta_1(y, x, beta_0, tau, hypers["mu_1"],␣
 ↪hypers["tau_1"])

        # Finally update tau with new beta_0 and new beta_1
        tau    = sample_tau(y, x, beta_0, beta_1, hypers["alpha"],␣
 ↪hypers["beta"])

        # Store the parameters sampled at every iteration
        store_param[i,:] = np.array((beta_0, beta_1, tau))

    store_param = pd.DataFrame(store_param)
    store_param.columns = ['beta_0', 'beta_1', 'tau']

    return store_param
```

[8]:
```
iters = 500

# Run the gibb's sampler for 500 iterations and get the trace of the parameters␣
 ↪sampled for plotting in the next cell
store_param = gibbs(y, x, iters, init, hypers)
```

[9]:
```
param_plot = store_param.plot()
param_plot.set_xlabel("Iteration")
param_plot.set_ylabel("Parameter value")
param_plot.grid()
```

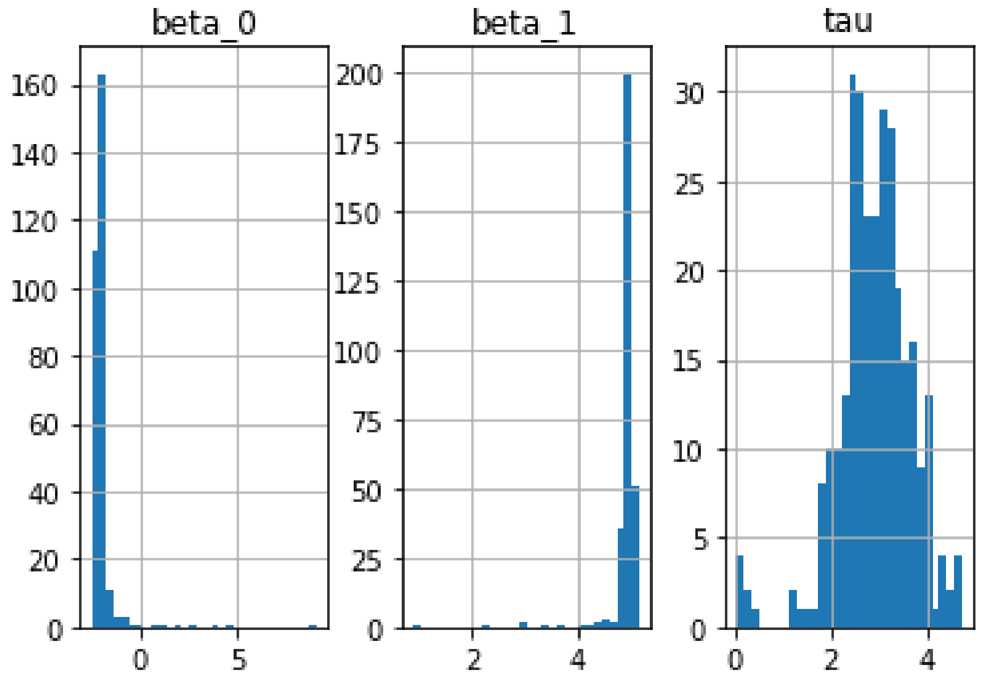


The results look great! **Congratulations!**

The true values for the toy model were $\beta_0 = -2$, $\beta_1 = 5$, $\tau = 3$. From the plot above we observe that after 20-30 iterations the sampler is already close to true result. The number of iterations needed for the sampler's exploration to take place and to start converging to the true posterior parameters is called the burn-in period. Let us look at the statistics of the parameters after this burn-in period.

[10]:
```
# Stats and histogram of all parameters after 300 iterations (or burn-in)

print("Parameter means")
print(store_param[:-200].median())

print()

print("Parameter standard deviations")
print(store_param[:-200].std())

print()

hist_plot = store_param[:-200].hist(bins = 30, layout = (1,3))
```

```
Parameter means
beta_0   -1.983210
beta_1    4.950847
tau       2.943861
dtype: float64

Parameter standard deviations
beta_0    0.951552
beta_1    0.361782
tau       0.764096
dtype: float64
```



We have now predicted the posterior parameter values to under a standard deviation error.

So, here we have successfully finished Bayesian Linear Regression using the Gibb's sampling approach.

I also have more on Gibb's sampling - **Inferring Gaussian Mixture Models (GMM) parameters using Gibb's sampling**.